Université de Lille — Faculté des Sciences et Technologies, Département Informatique

ASD

février 2018
Licence Informatique - Semestre 4

DS1 - documents de cours, TD, TP autorisés - durée 2h

Note : lorsque, dans une question, est demandée la complexité, c'est une fonction de la taille de la donnée qui est attendue. Lorsque c'est le comportement asymptotique, la réponse attendue est soit une notation $\mathcal{O}$, soit une notation Θ soit une notation Ω (le choix vous incombe).

Vous pourrez utiliser toute fonction vue en cours, TD, TP à la condition expresse de spécifier correctement les entrées et les sorties.

Le barême est donné à titre indicatif.

Livres, calculatrices et objets de communication interdits.

Exercice 1 : Questions de cours [2 points] (réponses sans justification)

Q 1.1 Si je prouve qu'un algorithme est en $\Theta(n^2)$, peut-il être en $\Omega(n^3)$? Répondre par oui ou non.

Q 1.2 Si je prouve qu'un algorithme est en $\Theta(n \log n)$ dans le pire des cas, peut-il exister des cas en $\Omega(n^2)$? Répondre par oui ou non.

Q 1.3 On peut résoudre l'équation de récurrence (qui compte les appels de `f`) de l'algorithme suivant en utilisant le théorème général. Répondre par vrai ou faux.

```
def f(t):
  if t == []:
    return 0
  else:
    return 1 + 2 * f(t[0:len(t)//4])
```

Q 1.4 $c(n) = 5c(\frac{n}{5}) + \frac{n}{2} = \Theta(n \log n)$. Répondre par vrai ou faux.

Exercice 2 : Complexité [5 points]

Compétence évaluée : calculer des complexités, définir pire et meilleur des cas.

On donne l'algorithme suivant, qui prend en entrée un tableau `t` de n `Element` (la classe `Element` est rappelée à la fin de l'énoncé).

On suppose que n est pair.

```
    n = len(t)
    s1 = Element(0)
    s2 = Element(0)
    fini = False
    i = 0
    while i <= n // 2 and not fini:
        s1.add(t[i])
        s2.add(t[n-1-i])
        if s1.cmp(s2) == 0:
            fini = True
        i = i + 1
```

Q 2.1 Décrire un meilleur des cas pour le nombre d'**additions** d'`Element` (le nombre d'appels à la méthode `add`).

Q 2.2 Donner exactement le nombre $c_m(n)$ d'additions d'`Element` pour un tableau de taille n pour le meilleur des cas.

Q 2.3 Décrire un pire des cas pour le nombre d'**additions** d'`Element`. Puis donner un exemple pour $n = 10$.

Q 2.4 Donner exactement le nombre $c_p(n)$ d'additions d'`Element` pour un tableau de taille n pour le pire des cas. Justifier.

Q 2.5 Donner le comportement asymptotique de cet algorithme pour le nombre d'additions d'`Element`.

Q 2.6 Donner le comportement asymptotique de cet algorithme pour le nombre de comparaisons d'`Element` (le nombre d'appels à la méthode `cmp`). Justifier.

Exercice 3 : Revisite du tri bulle [6 points]

Compétence évaluée : comprendre un nouvel algorithme, analyser des résultats expérimentaux.

L'algorithme du tri bulle est souvent présenté avec une double boucle pour aboutissant à une complexité en nombre de comparaisons en $\Theta(n^2)$ (comme vu en cours).

Voici une version un peu modifiée :

```
def bubbleSort(t):
    """
    :param t: numpy array of Element
    """
    fini = False
    indices = range(len(t)-1)
    while not fini:
        echange = False
        for i in indices:
            if cmp(t[i],t[i+1]) > 0:
                t[i], t[i+1] = t[i+1], t[i]
                echange = True
        if not echange:
            fini = True
```

Q 3.1 Décrire un meilleur des cas en nombre de comparaisons (le nombre d'appels à la fonction `cmp`).

Q 3.2 Quel est le nombre exact de comparaisons dans le meilleur des cas.

Q 3.3 Décrire un pire des cas.

Q 3.4 Quel est le nombre exact de comparaisons dans le pire des cas.

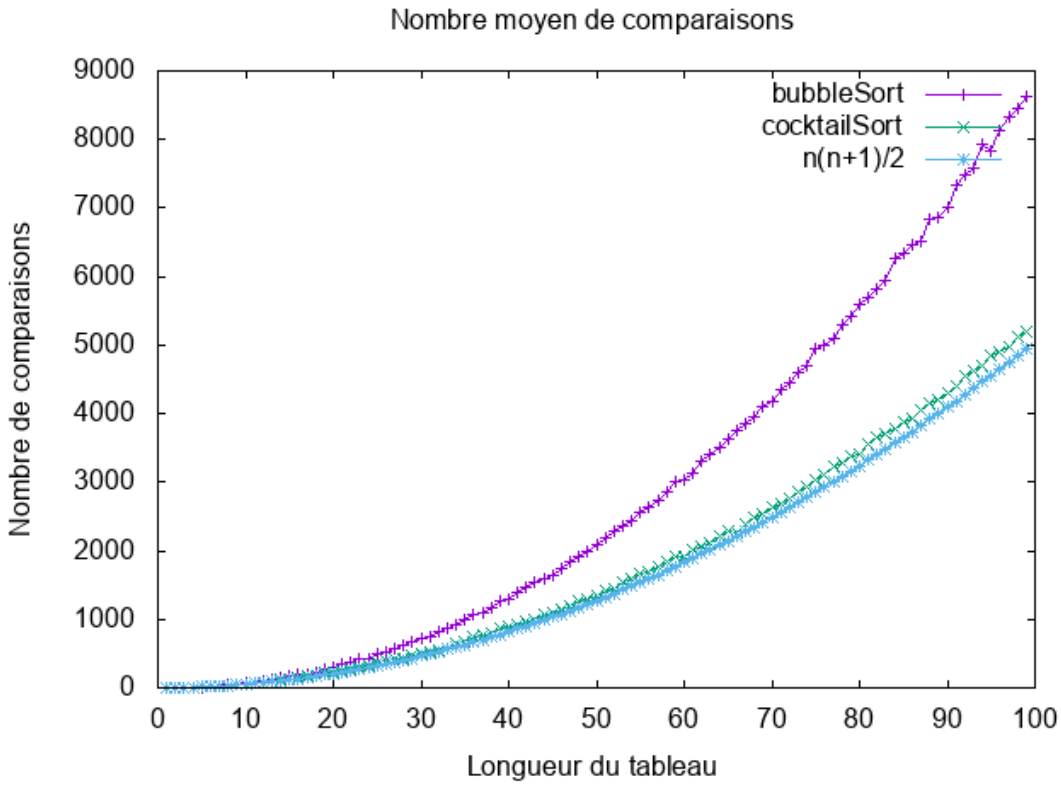
On propose maintenant une amélioration [1] :

```
def cocktailSort(t):
    fini = False
    indices = range(len(t)-1)
    while not fini:
        echange = False
        for i in indices:
            if cmp(t[i],t[i+1]) > 0:
                t[i], t[i+1] = t[i+1], t[i]
                echange = True
        if not echange:
            fini = True
        if not fini:
            echange = False
            for i in reversed(indices):
                if cmp(t[i],t[i+1]) > 0:
                    t[i], t[i+1] = t[i+1], t[i]
                    echange = True
            if not echange:
                fini = True
```

Q 3.5 Le meilleur des cas est-il modifié ? Expliquez pourquoi.

Q 3.6 Le pire des cas est-il modifié ? Expliquez pourquoi.

La courbe ci-dessous montre le nombre moyen de comparaisons réalisé sur un tirage aléatoire de 100 tableaux pour chaque longueur entre 1 et 100, pour le bubbleSort et pour le coctailSort.



Q 3.7 Qu'en conluez-vous sur l'efficacité du cocktailSort par rapport au bubbleSort ?

1. On trouve différentes dénominations de ce tri : cocktailSort ou shakerSort

Q 3.8 Donner un exemple d'un tableau de taille 5 où le coctailSort est plus efficace (vous déroulerez les étapes de chacun des deux tris en affichant le contenu du tableau après chaque boucle pour).

Exercice 4 : Calcul des deux points les plus proches [7 points]

Compétence évaluée : comprendre un algorithme décrit en langage naturel, établir l'équation de récurrence donnant la complexité d'un algorithme récursif, appliquer le théorème général.

Nous souhaitons concevoir un algorithme qui, étant donné un ensemble de points dans le plan, est capable de trouver les deux points les plus proches (au sens de la distance euclidienne).

Un algorithme naïf consiste à prendre chaque point, puis à calculer sa distance par rapport aux autres points, et à conserver les deux points dont la distance est minimale.

Les points seront représentés par des couples en Python et l'ensemble des points sera représenté par une liste. On suppose donnée une fonction `distance` qui prend deux points en entrée et calcule la distance qui les sépare en temps constant.

Q 4.1 Ecrire en Python la fonction suivante :

```
def point_les_plus_proches_naif (l):
    """
    Calcule les deux points les plus proches de la liste l.

    :param l: liste des points
    :type l: List de couples de float

    :return: les deux couples correspondant aux points les plus proches

    CU: la liste contient au moins 2 points
    """
```

Q 4.2 Quel est le nombre de calculs de distances entre deux points réalisé par votre algorithme naïf donné à la question précédente si l'ensemble des points est de taille n ?

On présente maintenant une autre stratégie. On ne vous demande pas de comprendre pourquoi cela fonctionne dans le cadre de cet examen, car la preuve de l'algorithme n'est pas évidente.

On supposera que la liste des points ℓ est triée selon les abscisses des points (cela ne rentre pas en compte dans la complexité de l'algorithme). Puis on applique l'algorithme suivant récursivement :

Etape 1 diviser ℓ en deux : ℓ^1 contient les $\lfloor n/2 \rfloor$ premiers points [2], ℓ^2 contient les $\lceil n/2 \rceil$ derniers points

Etape 2 trouver récursivement les deux points les plus proches dans ℓ^1 et dans ℓ^2, notons respectivement d_1 et d_2 les distances de ces deux paires de points, on note $d = \min(d_1, d_2)$

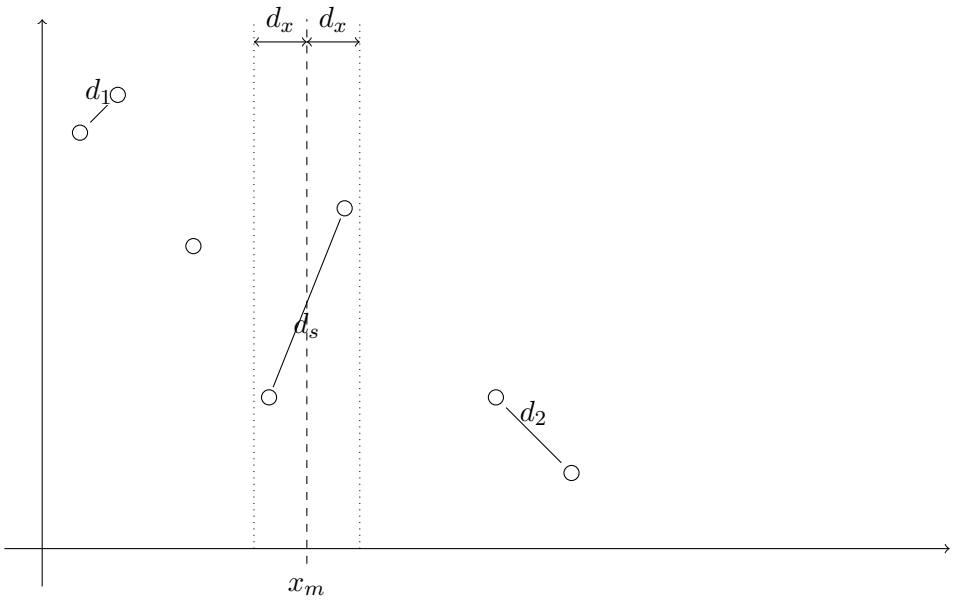
Etape 3 on considère l'abcisse $x_m = \frac{\ell^1[-1][0]+\ell^2[0][0]}{2}$, construire la liste s des points de ℓ dont les abscisses sont au maximum à une distance d de l'abscisse x_m

Etape 4 trier s selon l'axe des ordonnées

Etape 5 trouver les deux points les plus proches dans s (**en utilisant l'algorithme naïf**). Soit d_s la distance entre ces points. Si s est vide, $d_s := d$.

Etape 6 retourner le minimum entre d et d_s

2. ceux d'abscisse la plus faible étant donné que ℓ a été triée selon les abscisses



Q 4.3 Quel nom donne-t-on à la stratégie mise en place dans cet algorithme ?

Q 4.4 Quel tri choisir pour les tris de listes dans l'algorithme ? Justifier ce choix par des considérations sur la complexité de votre algorithme.

Q 4.5 L'algorithme décrit n'indique pas quand la récursion s'arrête. Décrire en français le ou les cas de base.

Q 4.6 Quel est le nombre d'appels à la fonction distance à l'étape 5 dans le pire des cas ? Justifier en décrivant le pire des cas.

Q 4.7 Donner l'équation de récurrence de l'algorithme récursif dans le pire des cas ?

Q 4.8 En déduire la complexité asymptotique de l'algorithme récursif dans le pire des cas. Justifier.

En cherchant à déterminer le meilleur des cas, un étudiant propose ceci : « *Pour l'étape 5 le meilleur des cas est lorsque l'ensemble s est vide, il n'y a jamais de calcul de distance et la complexité de l'algorithme de recherche des deux points les plus proches dans le meilleur des cas est* $c(n) = 0$ ». Ses camarades lui indiquent qu'il se trompe.

Q 4.9 Quelle erreur de raisonnement a commis cet étudiant ?

```
import time
from functools import total_ordering

@total_ordering
class Element:

    def __init__(self,value):
        assert(type(value) == int)
        self.value = value

    def __add__ (self, other):
        return Element(self.value + other.value)

    def __eq__(self, other):
        return self.value == other.value

    def __ne__(self, other):
        return not (self == other)

    def __lt__(self, other):
        return self.value < other.value

    def __repr__(self):
        return "{}".format(self.value)
```